Remarks

I. Interview with the Examiner

The Applicant thanks the Examiner for his time and courtesy in a telephone interview on 1/23/2006.

The Examiner asked that the term "cache directory address" be used in the claims to refer to a particular row in the cache directory. The Examiner felt that since addresses used in the specification were directed to accessing portions of the cache directory that use of "cache directory address" instead of simply "address" is warranted and serves to clarify the claims.

The Examiner was uncertain of whether the term "casting out" was well known in the art. The Examiner and Applicant agreed to look for public use of the term.

The Examiner felt that "locking" and "unlocking" was clear and supported regarding timer counter, the first refill counter and the second refill counter, or, alternatively, Applicant could amend claims to use the terms "inhibited" and "uninhibited" which were the words used in a specification in which those elements are described as apparatus.

In claims 3, 4, the Examiner requested that those claims be clarified that when a first portion is increased in size, a second portion is reduced in size. Applicant agreed to clarify claim 3.

II. Status of the Specification

Applicant has discovered several reference numbering errors that are being corrected in this response. Paragraphs [0056], [0057], and [0058] in the specification are amended to correct these errors, and replacement sheets for sheets 4 and 5 are attached with correct reference numbers.

Reference number 50 is used for both "snoop directory entry" (Figs. 4 and 5) and "Timer Counter 50" (Fig. 2). "Snoop directory entry 50" will be re-referenced as "snoop directory entry 4".

Reference numeral 55 refers to both "remote memory directory entry 55" (Fig. 2) and "remote memory directory entry 55" shown in Figs. 4 and 5. "Remote memory directory entry 55" will be re-referenced as "remote memory directory entry 5".

Reference number 51 is used for both "snoop directory control field 51" (Fig. 5) and for "timer control value 51" (Fig. 2). "Snoop directory control field 51" will be re-referenced as "snoop directory control field 6".

Reference number 52 in Fig. 5 points to "an entry type identifier 51A" (paragraph [0056])". Reference number 52 is also used for "compare 52" in Fig. 2. Use of identifier "51A" would cause confusion with "timer compare value 51" (Fig. 2). Therefore, Reference number 52 in Fig. 5, and paragraph [0056] are amended to identify the "entry type identifier 51A" as "entry type identifier 7".

Reference number 56 is used for both "remote memory directory control field 56" (Fig. 5 and paragraph [0056]) and for "allocation control 56" (Fig. 2). "Remote memory directory control field 56" is amended in the specification and Fig. 5 to be "Remote memory directory field 8".

Reference number 57 is used for both "entry type identifier 56A" (Fig. 5 and paragraph [0056]) and for "allocation 57" (Fig. 2). "Entry type identifier 56A" is amended in the specification and Fig. 5 to be "entry type identifier 9".

In paragraph 3 of the Office Action the Examiner states: "The specification is objected to as failing to provide proper antecedent basis for the claimed subject matter. ... Correction of the following is required: Specification does not provide sufficient antecedent basis for removing an address range and removing a segment in claims 5 and 6, respectively, casting out a snoop directory entry in claim 5, or locking and unlocking the timer counter, the first refill counter, and the second refill counter in claim 13."

As noted above, regarding "address range" in the interview, the Examiner's concern (voiced in paragraph 8 of the Office Action) is directed at vagueness whether the term "address range" pertains to addresses of rows in the cache directory or whether "address range" pertains to main memory locations". In the interview, Applicant agreed to clarify claims 5 and 6 by amending Claims 5 and 6 to use the term "cache directory address range". Applicant believes these amendments to claims 5 and 6 overcome the Examiner's objection to claims 5 and 6 in paragraph 3 of the Office Action.

Applicant investigated the term "castout" (cast out, cast-out), and believes that the term is quite commonly used for a victim cache line that is to be written over by a new cache line, and which must be written back (cast out) to a higher level of cache, or main store. There are numerous uses of the term in US patent literature, for example, US 6,195,729, "Deallocation with cache update protocol (L2 evictions)", which includes the sentence, "The process of writing modified data from a victim to system memory or a lower cache level is called a cast out or eviction."

US 5,809,536 "Method for reducing the number of coherency cycles within a directory-based cache coherency memory system utilizing a memory state cache" includes: "CASTOUT INVALIDATE (MBCOI) This cycle is initiated to cause a cache with a potentially modified copy of an addressed line to cast it out to system memory and to go to the invalid state (PI)."

US 6,163,857 "Computer system UE recovery logic" includes: "The line containing the bad data is checked to see if it was changed, 206. If changed, the data must be cast-out to the main store,..".

In non-patent literature, ARM926EJ-S Technical Reference Manual copyright 2001-2003 ARM Limited, on page Glossary-18, states (for "Victim"): "A cache line,

selected to be discarded to make room for a replacement cache line that is required as a result of a cache miss. The way in which the victim is selected for eviction is processor-specific. A victim is also known as a cast out." Applicant attaches a copy of the front page of ARM926EJ-S, and page Glossary-18 at the end of this paper.

In "IBM Journal of Research and Development", Jul-Sep 1997 by Mak, P, et.al, "Shared-cache clusters in a system with a fully shared memory" (copy of journal cover and cited article attached at the end of this paper) there are several instances of "cast-out". For example, on page 435, "For a cast-out operation (either LRU or XI), this information is used to determine whether the copy of the line in this L2 cache is more recent than the copy in the main memory. If so, a memory update operation and/or XI cast-out operation will result." For a second example, on page 439, "The LSAR controller (an LRU cast-out is required." Or, on page 441, "Each time a fetch operation from a processor misses the L2 cache, LSAR is responsible for the eviction of the LRU targeted line from the cache to make room for the new fetch-miss data. In an arrangement analogous to the LFAR just described, there are two LSAR registers, LSARA and LSARB, dedicated to each BSN port, sharing a single line store buffer. The cast-out data are held in the buffer until they are allowed to transmit on the bus."

The above examples are just a few of the public uses of "cast out", and Applicant believes the term is sufficiently clear and public to use in the present application. Therefore, Applicant respectfully requests that the Examiner withdraw his objection (paragraph 3 of the Office Action) to the term.

In paragraph 3, the Examiner objected to use of the terms "locking and unlocking the timer counter, the first refill counter, and the second refill counter in claim 13." Applicant believes that paragraph [0062] provides sufficient antecedent: "If, however, the time interval has elapsed, control passes to step 64. In step 64, the timer counter is locked, preventing further counting. The two refill counters are also locked, preventing further accumulation of snoop directory entry refills and remote memory directory refills." Applicant acknowledges that the word "inhibiting" or "inhibits" was used in paragraphs [0040]-[0042] but believes that "locked" is synonymous and the term "locked" is used in paragraph [0062]. Applicant therefore respectfully requests that the

Examiner reconsider his objection to the terms "locking and unlocking" in paragraph 3 of the Office Action.

In paragraph 3 of the Office Action, the Examiner objects to paragraph [0002] of the specification, asking the Applicant to "update status of case 10/403,157". In response, Applicant amends paragraph [0002] to include the publication number, US20040193810A1 of the related invention.

II. Status of the Claims.

In the Office Action, the Examiner indicated that claims 1-18 are pending, 1, 5, 7-9 and 18 are allowed. Claims 3-4, 6, and 17 are rejected. Claims 2 and 10-16 are objected to.

In paragraph 4 of the Office Action, the Examiner states: "Claims 2-4 and 10-16 are objected to because of the following informalities:..." The Examiner objects to Claim 2 "because of the following informalities: Line 1 of claim 2 reads, '...further comprising comprising the steps...' Use of 'comprising' here is repetitive." In response, Applicant amends Claim 2 to correct the repetitive use of "comprising". Applicant believes the amendment to Claim 2 also overcomes the same objection to claims 3 and 4 which depend from Claim 2. The Examiner further states that "Claim 10 is objected to because of the following informalities: Line 8 reads '...control capable making a...' Examiner suggests '...control capable of making a...'" In response, Applicant amends Claim 10 according to the Examiner's suggestion. Applicants believe the amendment to Claim 10 makes Claim 10 and its dependent claims 11-16 allowable.

In paragraph 6 of the Office Action, the Examiner rejects claim 3 and dependent claim 4 under 35 U.S.C. 112. "Finiteness of cache memory precludes increasing both the first and second portion simultaneously. Claim 3 depends from claim 2 which claimed increasing the first portion and the second portion under particular conditions intended to be mutually exclusive, that is, make the first portion bigger when there are a relatively larger number of snoop directory entry refills, and make the second portion bigger when there are a relatively larger number of remote directory entry refills" (see [0063], for example). Applicant amends Claim 2 to clarify that when the first portion is increased, the second portion is decreased, and when the second portion is increased, the first portion is decreased. Claim 2 was not rejected in paragraph 6, but Applicant believes that Claim 2 should be clarified, as well as claim 3. Applicant also amends Claim 3 to make the same clarification. Support for increasing one portion and at the same time decreasing the other portion is also found in paragraph [0046], "If an undesirably high number of snoop directory entry refills occurs relative to the number of remote directory entry refills occurs during a time interval, more space is allocated to snoop directory 46, and less space is allocated

to remote memory directory 47." Applicant believes that the above amendments to Claims 2 and 3 suffice to make Claim 4 allowable. Applicant believes that the amendments overcome the rejections under 35 U.S.C. 112, and respectfully request the Examiner to reconsider and withdraw the rejection.

In paragraph 8 of the Office Action, the Examiner states "Claim 6 rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Claim 6 recites, '...removing a segment of a current address range of the first portion of the directory memory...' This is ambiguous in that it is unclear if the address range is a cache address range to be reallocated or if the address range refers to main memory locations to be remapped into new cache locations." As discussed above, Applicant has amended Claims 5 and 6 to clarify that it is the cache directory address range that is being reallocated. Applicant respectfully asks that the Examiner reconsider and remove his rejection of Claim 6 under 35 U.S.C. 112, second paragraph.

In paragraph 9 of the Office Action, the Examiner rejects claim 17 under 35 U.S.C. 101 "because the claimed invention is directed to non-statutory subject matter. A computer program product not embodied in a computer readable medium is unpatentable subject matter. In response, Applicant amends claim 17 to include a limitation that the computer program product is embodied in a computer readable medium. Applicant believes that paragraph [0065] supplies sufficient support: "the program product can exist on and be distributed on media that can be read by and executed by a suitable computer. Such media include but are not limited to CDROM disks, floppy disks, hard disks, and magnetic tape." Applicant respectfully asks the Examiner to reconsider Claim 17 as amended and withdraw the rejection under 35 U.S.C. 101.

III. Allowable Subject Matter

In paragraph 10 of the Office Action, the Examiner states: "Claims 1-2, 5, 7-16, and 18 are allowable over the prior art. Claims 2 and 10-16 are objected as set forth above." Applicant thanks the Examiner for the allowances.

V. Conclusion

In view of the foregoing comments and amendments, Applicant respectfully requests that the application, with claims 1-18, be passed to issue.

Respectfully submitted,

JOHN M. BORKENHAGEN

By: Robert R Williams

Robert R. Williams, Patent Agent
Registration No.: 48,395
IBM Corporation - Department 917
3605 Highway 52 North
Rochester, Minnesota 55901-7829

Telephone: (507) 253-2761
Fax No.: (507) 253-2382

ARM926EJ-S

(r0p4/r0p5)

Technical Reference Manual

ARM

ARM DDI0198D

Unpredictable For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.

VA *See* Virtual Address.

Victim A cache line, selected to be discarded to make room for a replacement cache line that is required as a result of a cache miss. The way in which the victim is selected for eviction is processor-specific. A victim is also known as a cast out.

Virtual Address (VA)

The MMU uses its page tables to translate a Virtual Address into a Physical Address. The processor executes code at the Virtual Address, which might be located elsewhere in physical memory.

See also Fast Context Switch Extension, Modified Virtual Address, and Physical Address.

Warm reset Also known as a core reset. Initializes the majority of the processor excluding the debug controller and debug logic. This type of reset is useful if you are using the debugging features of a processor.

Watchpoint A watchpoint is a mechanism provided by debuggers to halt program execution when the data contained by a particular memory address is changed. Watchpoints are inserted by the programmer to allow inspection of register contents, memory locations, and variable values when memory is written to test that the program is operating correctly. Watchpoints are removed after the program is successfully tested. *See also* Breakpoint.

Way *See* Cache way.

WB *See* Write-back.

Word A 32-bit data item.

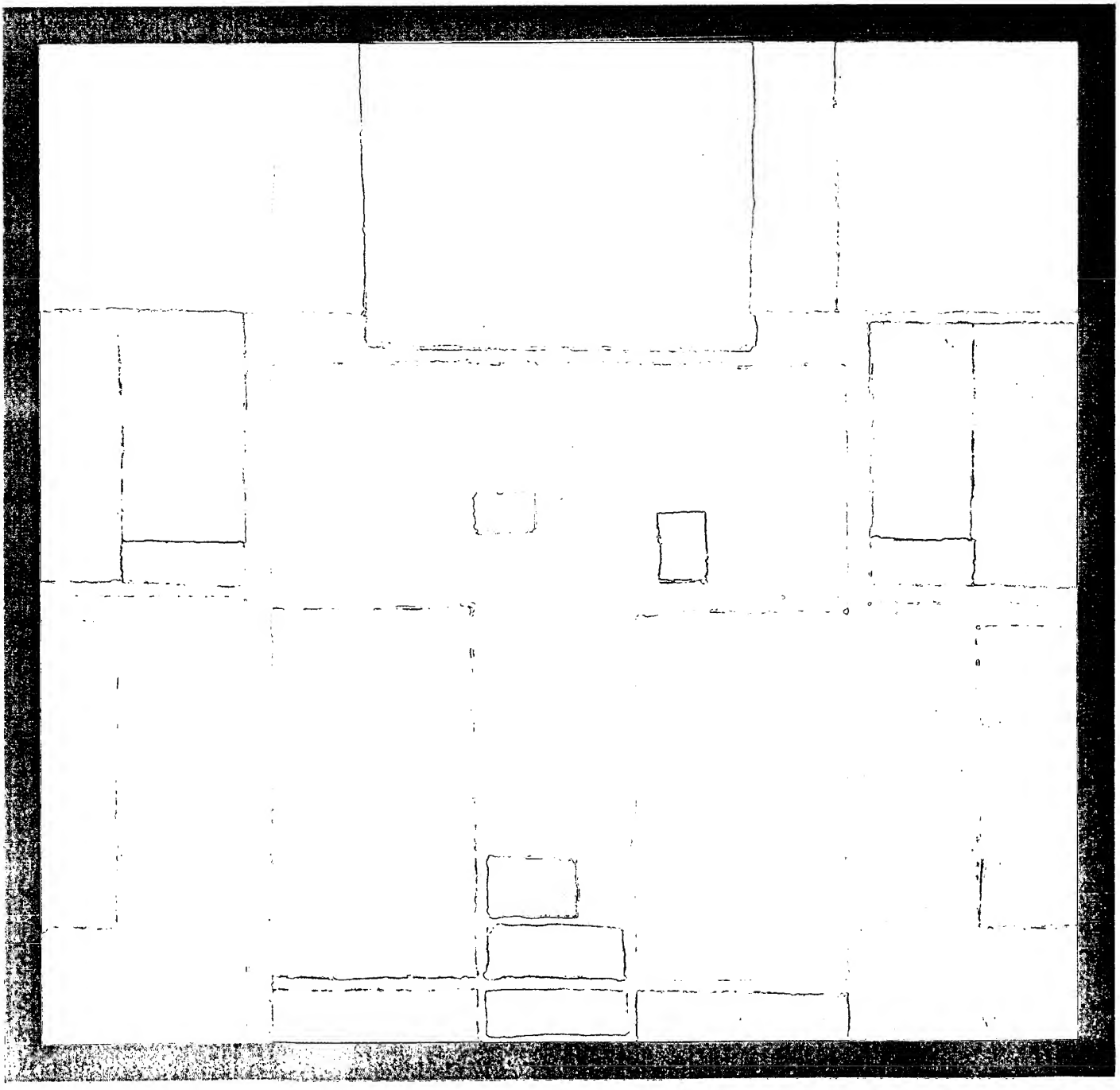
Write Writes are defined as operations that have the semantics of a store. That is, the ARM instructions SRS, STM, STRD, STC, STRT, STRH, STRB, STRBT, STREX, SWP, and SWPB, and the Thumb instructions STM, STR, STRH, STRB, and PUSH. Java instructions that are accelerated by hardware can cause a number of writes to occur, according to the state of the Java stack and the implementation of the Java hardware acceleration.

Write-back (WB) In a write-back cache, data is only written to main memory when it is forced out of the cache on line replacement following a cache miss. Otherwise, writes by the processor only update the cache. (Also known as copyback).

Write buffer A block of high-speed memory, arranged as a FIFO buffer, between the data cache and main memory, whose purpose is to optimize stores to main memory.

Journal of Research and Development

Volume [illegible] Number [illegible] [illegible] 1957



Shared-cache clusters in a system with a fully shared memory

by P. Mak
M. A. Blake
C. C. Jones
G. E. Strait
P. R. Turgeon

Interest in the concept of clustered caches has been growing in recent years. The advantages of sharing data and instruction streams among two or more microprocessors are understood; however, clustering also introduces new challenges in cache and memory coherency when system design requirements indicate that two or more of these clusters are needed. This paper describes the shared L2 cache cluster design found in the S/390® G4 server. This novel cache design consists of multiple shared-cache clusters, each supporting up to three microprocessors, forming a tightly coupled symmetric multiprocessor with fully coherent caches and main memory. Because this cache provides the link between an existing S/390 system bus and the new, high-performance S/390 G4 microprocessor chips, the paper addresses the challenges unique to operating shared caches on a common system bus.

Introduction

As the S/390* transformation from its legacy of large, complex, bipolar-based mainframes to simpler, CMOS-microprocessor-based servers continues, increasing emphasis is being placed on the system structure and memory hierarchy. As described by Getzlaff et al. [1], the S/390 Parallel Enterprise Server Generation 3 (G3) features a system structure which supports microprocessor and system frequencies in excess of 160 MHz. This design features conventional, dedicated L1 and L2 caches as well as a novel, shared L2.5 cache [2] which services all of the microprocessors in the system. This topology provides impressive system-level performance and has excellent scaling capabilities.

However, the introduction of the S/390 G4 microprocessor [3], with its design goal of doubling the microprocessor frequency, was expected to increase memory subsystem performance requirements. The L2 cache design team was challenged to provide the means to enable the introduction of this new microprocessor while reusing as many S/390 G3 components as possible. The first challenge involved system cycle time. Because of the desire to reuse the S/390 G3 memory hierarchy, speed matching between the new, faster S/390 G4 microprocessor and the reused components would be needed. Second, the utilization level of the system bus was observed to be increasing rapidly with increasing microprocessor frequency. Doubling the microprocessor speed was likely to cause considerable acceleration of this effect, thereby necessitating a mechanism for reducing bus traffic. Third, analysis of typical S/390 workloads indicated that there was a large degree of sharing of instructions and data among the processing elements of an S/390

0018-8646/97/$5.00 © 1997 IBM

system. It would thus be very advantageous to introduce a shared-cache structure to the cache hierarchy. Finally, during the design of the S/390 G3 system, it became apparent that a large symmetric multiprocessor introduced unwanted packaging complexity on the bus connections among the various system components. This last challenge was solved on the S/390 G3 system by distributing four microprocessors' private L2s across four SRAM cache chips [1].

A number of alternative designs were considered for the S/390 G4 L2 cache. The initial desire was to introduce a large, fully shared cache such as those found in bipolar ES/9000* SMPs. While this would certainly have provided optimal system-level performance via higher cache efficiency, this design approach was inconsistent with the existing system bus design. In particular, the S/390 G3 memory interface would have required a complete redesign to accommodate a fully shared L2 cache. Performance analysis, however, indicated that introducing cache sharing on each of the existing BSN system buses would generate a large percentage of the advantage of shared caches while also reducing system bus traffic. This scheme does not disrupt the system design and bus protocols. The S/390 design team therefore developed a hybrid solution consisting of fully shared three-way cache clusters which attach to the existing S/390 G3 system bus [4].

This L2 cache design resulted in an amount of aggregate L2 equivalent to that available in the S/390 G3 (3 MB of total L2 capacity) while reducing the L2 chip count by 33%. The value of shared caching was confirmed by performance measurements which showed that the shared caches improved cache hit rates by 40–50% over the predecessor system. The cross-point switches introduced at the CP interface maximize CP performance and minimize overall latencies, yet the system bus interface was also maintained, enabling component reuse. Finally, despite the fact that a shared-cache design is inherently more complex than private L2s, the entire design cycle, from design concept through initial product prototype, required only 15 months. Initial versions of the S/390 G4 L2 chip were fully functional and exceeded frequency goals; virtually the entire manufactured chip distribution operates at or better than 200 MHz in CMOS 5X, making this a very high-yield, low-cost, high-performance L2 cache chip.

The S/390 Parallel Enterprise Server Generation 4 memory hierarchy is depicted in **Figure 1**. Each microprocessor, or CP, contains 64 KB of store-through L1 cache. The L2 cache contains 768 KB distributed in a clustered configuration across a pair of chips. An L2 cluster supports up to three CPs and provides access to the entire memory space. The L2 cache chip provides the speed-matching function required by the S/390 G4 CP, allowing the CP to run at twice the frequency of the remainder of the system. This enables the reuse of the S/390 G3 memory hierarchy (BSN, STC, etc.).

This paper discusses the operation and design of the shared-cache clusters found in the S/390 G4 system[1]; it is divided into four main sections. The design and layout of the S/390 G4 L2 cache are first described, with particular attention to design challenges unique to a shared structure. The next section addresses cache coherency issues unique to a shared-cache environment, and the solutions devised to address them. The third section discusses interlock controls in further detail. The paper concludes with a brief summary of the key attributes of the S/390 G4 shared L2 cache.

To maintain clear and uncluttered descriptions of the operations performed by the shared L2 cache, the following convention is used throughout the remainder of the paper. The terms *L2* and *L2 cache* are used to refer to a single chip in an S/390 G4 L2 cache cluster. When discussing operations which span the pair of cache chips, the terms *cluster* or *L2 cluster* are used. The reader should also keep in mind that since a cluster consists of two independent L2 cache chips, and there may be up to four clusters in the current design point, up to eight independent L2 cache chips may simultaneously be performing the bus operations described here. Finally, the four clusters which comprise a fully populated system all share the same common main memory.

Glossary of terms

BIDI: Bidirectional
BIF: BSN InterFace
BSN: Bus Switch Network—the chip which controls bus traffic in S/390 G3 and G4 systems
BSNAR: BSN interface Address Register
BUSRR: BUS Request Register
CFAR: CP Fetch Address Register
CL: Changed Line bit—indicates that the data in the tagged line have been altered and are therefore of more recent vintage than those stored in main memory
CLST: Conditional Line STore
CP: Central Processor or Central microProcessor
CPAR: CP interface Address Register
CPID: Central Processor ID
doubleword: Eight bytes
L1: Level-1 cache
L2: Level-2 cache
LFAR/LFARA/LFARB: Line Fetch Address Register—register(s) used to stage CP fetch requests
LSAR/LSARA/LSARB: Line Store Address Register—register(s) used to stage store requests to main memory

[1] For an example of independent research confirming the concept of shared-cache clusters, see B. A. Nayfeh, K. Olukotun, and J. P. Singh, "The Impact of Shared Cache Clustering in Small-Scale Shared-Memory Multiprocessors," *Proceedings of the Second International Symposium on High Performance Computer Architecture*, San Jose, CA, 1996, pp. 74–84.

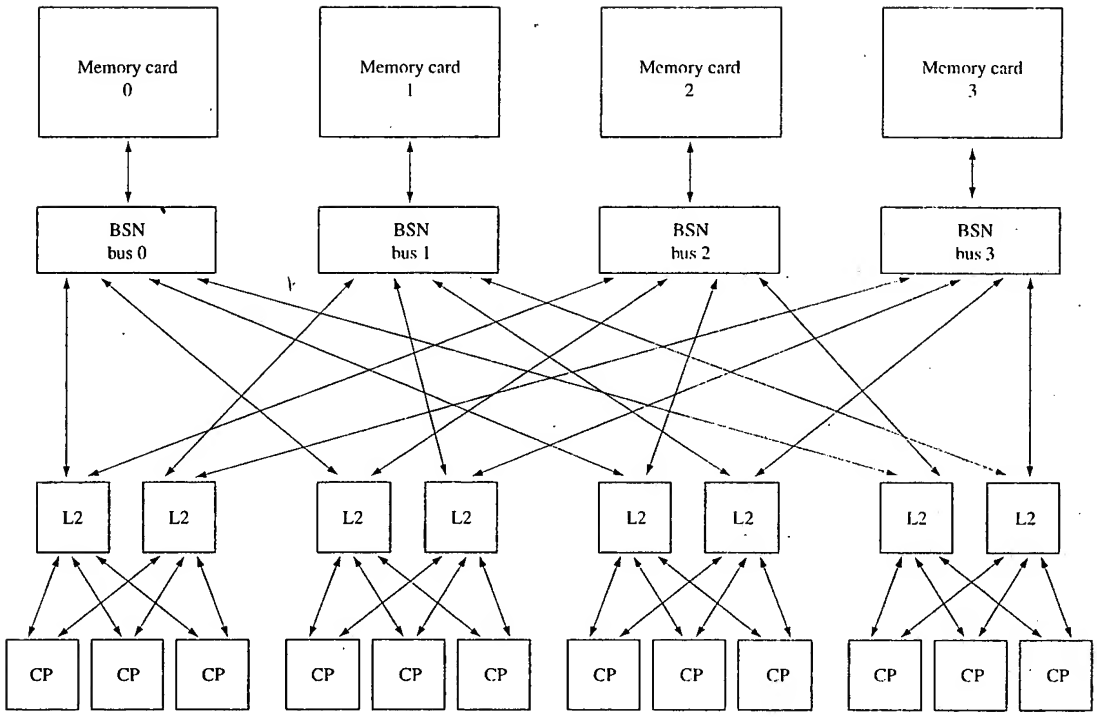


Figure 1

S/390 G4 memory hierarchy.

LSLF: Line Store/Line Fetch operation—a fetch with an accompanying cast-out

LRU: Least Recently Used—an algorithm for determining which line(s) to replace with newer lines in the cache

MC: Multi-Copy bit

quadword: Sixteen bytes

RAS: Reliability, Availability, Serviceability

SEC/DED: Single-Error Correct/Double-Error Detect ECC code

SMP: Symmetrical MultiProcessor

XI: Cross-Invalidate—a mechanism to ensure cache coherency by invalidating lines out of L1 and/or L2 caches

S/390 G4 shared L2 cache

The S/390 G4 shared L2 cache chip is an integrated design, with both the large custom SRAM [5] and cache controls contained on the same chip. The shared cache consists of two identical chips, each containing 384 KB of SRAM, for a total of 768 KB of cache per cluster. The entire memory address space is mapped across this pair of L2 chips, interleaved across the four memory cards to maximize concurrent operation. The cache itself is six-way set-associative, is dual-interleaved, and has directories dedicated to each system bus. Each L2 chip features nonblocking switch functions to each attached CP as well as to the other system components. Data buffers in the cache array itself are also provided in order to minimize latency effects. The result of the implementation of these design concepts is an L2 cache which enables concurrent execution of multiple requests, whether from the CPs or from other L2 clusters.

To enable achievement of CP cycle-time objectives, the L1 cache was designed as a store-through cache, meaning that altered data must also be stored to the next-level cache. The L2 cache is a store-in design, which helps to control and minimize system bus activity. A full subset cache protocol is used to manage data coherency between the L1s and L2s; this means that all lines contained in any L1 are also stored in that cluster's L2. The advantages of this scheme will become apparent below.

In addition to performance, the S/390 G4 L2 cache provides excellent RAS characteristics. This design contains SEC/DED ECC on both the cache and directory arrays. Error correction in all cases is provided "on the fly," with no additional access penalty. When uncorrectable errors are encountered, system recovery

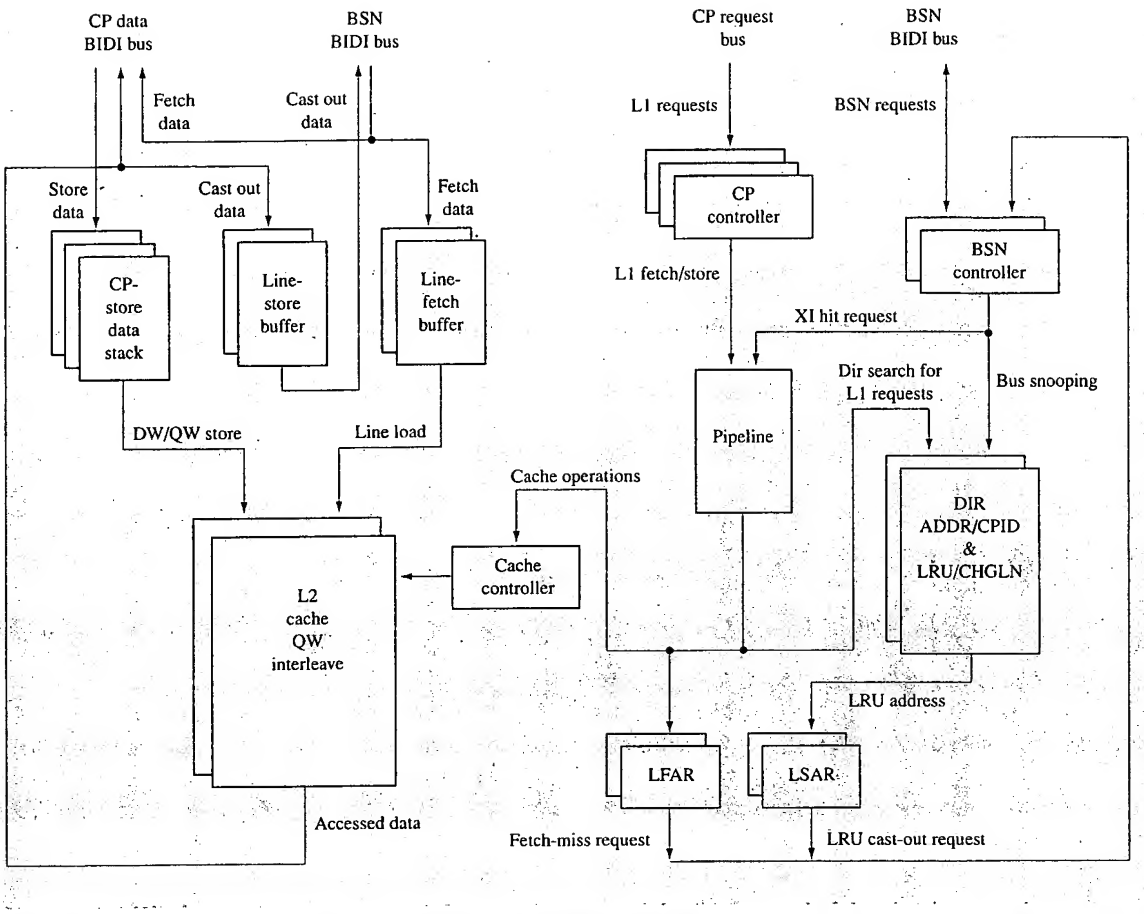


Figure 2

L2 data and address flow.

actions are triggered. The L2 cache and directory also support a "cache-line delete" function [6] which allows removal of a small number of cache lines from active use to prevent known stuck faults from becoming uncorrectable faults. Finally, in the S/390 G4 system, it is possible for one or more individual CPs to halt because of error without causing a system checkstop. Since cache coherency is managed by the shared L2, failing CPs can be logically removed from the active configuration with no data integrity exposure.

The high-level data and address flow of the S/390 G4 shared L2 cache is depicted in **Figure 2**. This shared L2 cache comprises the following critical component functions:

1. L2 pipeline and arbitration unit, which provide the means for controlling multiple requests for the shared L2 cache facilities.
2. L2 directory, which maintains the contents of the L2 cache array.
3. CP interface controller, which provides fetch/store access to the L2 from up to three CPs.
4. BSN interface controller, which supplies the interface to the other shared L2 clusters as well as to the shared L2.5 cache and main memory.
5. LSAR/LFAR controller, which stages and controls all fetch-miss and write-back activity.
6. Shared L2 cache array.

- *L2 pipeline*

The L2 pipeline is shown in **Figure 3**. All incoming requests processed in the shared L2 cache must pass through the central pipeline. This pipeline consists of a number of single-cycle staging registers that perform different actions each cycle. A new request can be launched into the pipeline every cycle; a request can take

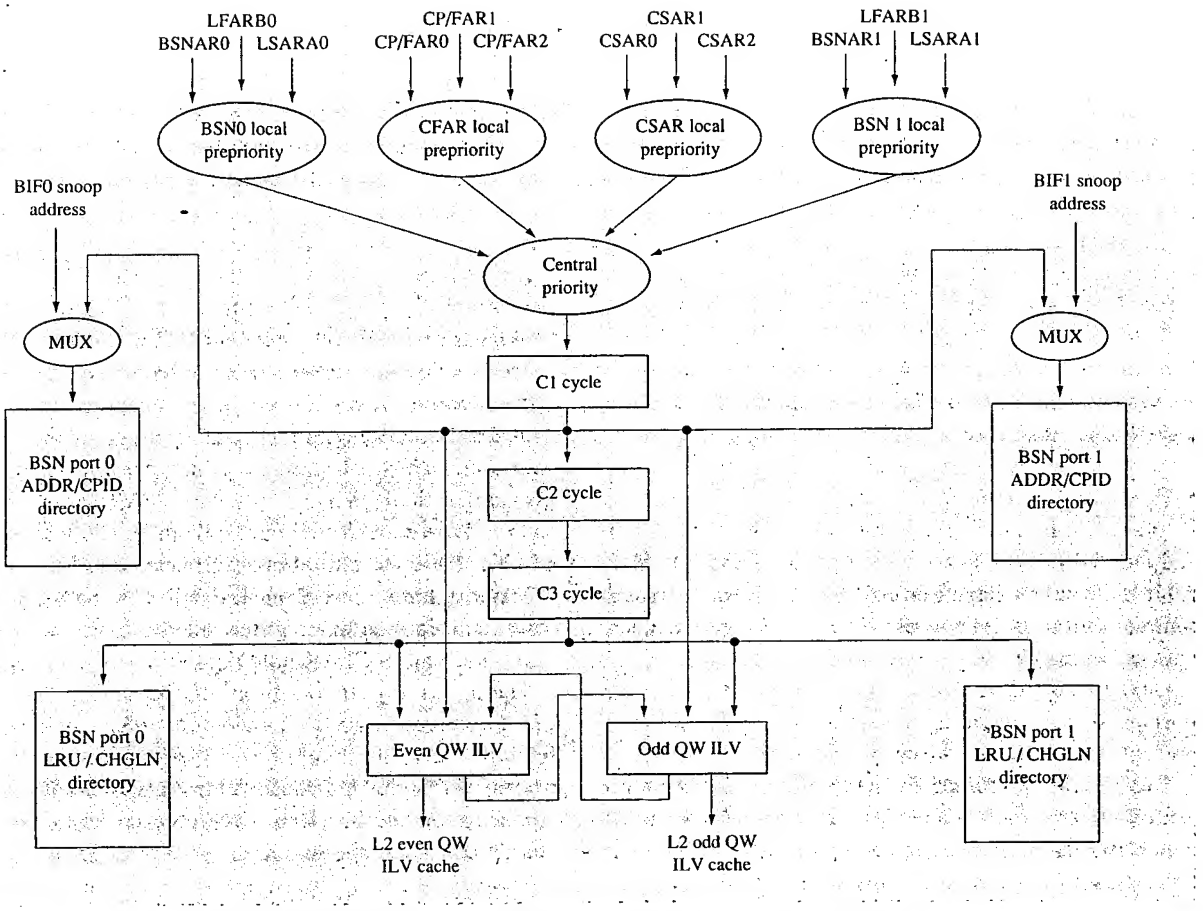


Figure 3

L2 pipeline.

as few as two cycles or as many as 11 cycles depending on its type and the availability of the resources needed for completion.

A pipelined design was selected in order to effectively manage the high level of activity from the L1 store-through traffic for the three CPs. Stores can degrade performance when they are not drained fast enough, thereby halting CP execution. To reduce the probability of creating a bottleneck, the L2 has a pipeline design which can process a new operation every cycle. The design also contains a store stack for each CP that is deep enough to buffer the stores until they can be processed through the pipe.

The central priority logic ultimately selects one requestor and gates the corresponding requestor information (such as address, command, mode, etc.) into the pipeline. A given requestor may make multiple passes through the pipeline, performing different functions on each pass. The action taken during each pipeline cycle depends on the type and source of the command, and on the results of previous passes through the pipeline.

- C1 cycle activities include
  - Directory lookup and field update
  - Command and address information sent to cache interleave controls
  - Address compares against internal resources and requestors performed
- C2 cycle activities include
  - Directory search results available
  - First interleave busy cycle for a cache access operation
  - LFAR facility selected on detection of L2 cache miss
  - L2 cache request proceed/reject, based on the resource and conflict resolution

- C3 cycle activities include
  - LRU array access and/or update
  - CL array access and/or update
- C4 cycle activities include
  - First interleave busy cycle for a cache write operation
  - Invalidate command sent to CP if the requested data have a coherency conflict
  - LSAR facility selected on L2 cache miss requiring a cast-out
  - Completion response sent to CP if requested data hit in the L2 cache with no coherency conflict

A cache read or write operation continues for several additional cycles beyond C4 to complete the transfer of eight 16-byte data shots.

- *Pipeline arbitration*

A total of 16 hardware requestors, dedicated for CP, BSN, and internal L2 facility requests, can vie for priority to enter the pipeline. Trying to route all 16 directly to some central arbitration logic would have constituted a major physical wiring challenge. Decentralizing the pipeline inputs by sending them through the prepriority arbitration stations before forwarding to central priority helped to prevent these potential wiring problems.

The prepriority arbiters funnel down the requests for central priority. Each selects one of its requestors and then forwards the request to central priority. At the same time, the prepriority arbiters forward the corresponding requestor ID tag, address, command, mode, etc. buses to the L2 pipeline. The prepriority arbiters also block requests whose resource needs are not available (e.g., cache interleave conflicts), and some arbiters equitably select requestors to avoid potential processing lockouts. This particular lockout avoidance and others are discussed in more detail in a later section.

There are a total of four prepriority stations within the L2 cache chip. Each BSN port has a dedicated station, while the three CP ports share the other two stations. A BSN prepriority station arbitrates among the BSNAR, LFARB, and LSARA functions dedicated to the particular BSN port. The selection is done in a priority order where BSNAR has the highest priority, LFARB is second, and LSARA is last. BSNAR is picked first because it reflects an XI (cross-invalidate) hit situation. During the time it takes an L2 to resolve an XI hit, the BSN bus is blocked from servicing a new operation. With bus utilization projected to be relatively high, a reduction of utilization and, hence, queuing was called for. LFARB and LSARA selection priority is not a critical performance issue because of the low projected L2 miss rate.

Since each CP can have both fetch and store operations pending at any point in time, two prepriority stations are supplied for the CP ports. Because fetch operations are more critical to system performance than stores, having separate arbiters for CP fetch and store operations ensures that a fetch will always be selected over a store whenever both operations are vying for priority concurrently. The CP fetch (or FAR) prepriority station arbitrates among requests from the three CP ports in a least-recently-processed scheme that avoids potential lockout. A normal round-robin or a priority order chain scheme can cause a lockout on a CP if there are sufficient activities from the other ports (such as when CPs are in synchronous loops). The-least-recently processed scheme is analogous to a least-recently-used (LRU) algorithm for cache-line management, in which a most recent tag is updated whenever a CP fetch (FAR) operation has completed processing in the L2. The inverse of this tag determines which CP's request is the oldest, thereby assigning it the highest priority in the FAR prepriority station. Even though there are only three CP ports on the L2 chip, there are a total of six request paths into this prepriority station, two from each CP. One path comes from the chip interface register called CPAR, and the other is from the CFAR hold register.

The prepriority station for arbitrating among the three CP store requests employs a modified round-robin scheme to allow one of the three store requestors to enter central priority [7]. Stores are checked for cache interleave conflict prior to arbitration in the prepriority station to guarantee completion in a single pass through the pipeline. Otherwise, pipe utilization would increase greatly because of wasted pipeline passes. One undesirable side effect of this greater pipeline efficiency is that some store requests can be locked out indefinitely if higher-priority operations such as CP fetches continue to keep the target cache interleave of the store operation busy. The modified round-robin selection scheme guarantees that if the highest-priority store request is not selected because of a cache interleave conflict, and a lower-priority store requestor in the round-robin is selected, the highest-priority store request will prevail as the first store to be selected in subsequent cycles. Normally in this situation, a regular round-robin would be updated and a new store requestor would be given the highest priority in the following cycle. However, in the modified scheme the round-robin is updated only when a store requestor is selected, and no higher-priority store requestors are blocked from priority because of cache interleave conflict.

After the initial filtering at the four prepriority stations, the surviving requestors meet in central priority for final selection into the pipeline for processing. At this station, a straightforward priority order chain selects the requestor. For reasons explained above, requestors from the two BSN prepriority groups have the highest priority, followed by the CP FAR prepriority group and finally by the store prepriority group. A grant is not always given whenever

there is a pipeline request. Some of the reasons relate to actions taken as part of recovery to prevent propagation of hardware faults and also to avoid certain classes of lockout problems caused by allowing into the pipeline requestors or operations that may inadvertently prevent another requestor or operation from completing processing. Requestors not granted pipeline access during the priority arbitration cycle continue to request priority during each machine cycle until a grant is received.

- *Directory*

Each L2 chip contains two logical directories (or cache tag arrays), one per BSN port. Each directory consists of 256 rows and is six-way set-associative. With a 128-byte cache-line size, the directory supports a total of 384 KB of L2 cache. A single logical directory contains the following fields: address, CPID, changed line (CL), and LRU. Physically, the address and CPID fields share the same custom SRAM macro, which is accessed or updated in the C1 cycle of an operation in the pipeline. The CL and LRU are contained in a shared SRAM array macro that is accessed or updated in the C3 cycle of the operation in the pipeline.

The address/CPID directory array may be accessed via the BIF register during a snoop search or via the C1 stage of the pipeline during other requests. The BIF register path to the directory is needed in order to maintain bus protocol with the BSN for reporting a possible XI hit on a bus snoop operation. Since there are two BSN ports on an L2 chip, the directory is logically partitioned into two halves to support the possibility of simultaneous directory lookups due to bus snoops on each BSN bus. Snoop requests may be received from the BSN ports at any time; when conflicts between snoops and other pipeline requests occur, the snoop receives priority, and the internal request is recycled through the pipeline once the snoop request has been satisfied.

The directory contains status information for data in the L2 cache, as well as information on data ownership. A cache line can exist in four possible ownership states: CP exclusive, CP read only with L2 exclusive, multiple copies in L2s, and invalid. This ownership information is in the three-bit CPID tag field, where one of the bits is the multiple copy (MC) tag bit.

The directory also maintains a changed line (CL) tag which tracks whether a given line has been updated in the L2 cache by a CP store operation or was brought into the L2 cache already in a changed state from another L2 cache via a BSN bus operation. For a cast-out operation (either LRU or XI), this information is used to determine whether the copy of the line in this L2 cache is more recent than the copy in main memory. If so, a memory update operation and/or XI cast-out operation will result. Each directory entry has a CL bit. Normally, the CL bit is set by a CP store operation in the C3 cycle of its pass through the pipeline when it is known whether or not the operation has been rejected.

An LRU array is also maintained for each congruence class (row) in the directory. The partitioned LRU algorithm determines which of the six compartments will be the one to be overwritten with new data when a new line is being written into the cache for that congruence class. If there are compartments which are not currently valid, one of these will be chosen first. If all compartments are valid, the compartment pointed to by the LRU code will be chosen. A partitioned LRU scheme was chosen because it can support a CP store operation every cycle and its smaller code field has a silicon area advantage over a true LRU algorithm, which requires a longer LRU code field. Also, comparisons of the partitioned and true LRU schemes have shown very little performance difference.

- *CP controller*

The CP controller is depicted in **Figure 4**. The CP controller logic provides the interface for up to three CPs, services all CP fetch and store requests, and handles XI requests and responses from the other L2 clusters. The interface contains a fetch/store command bus, a fetch/store address bus, a bidirectional fetch/store data bus, a fetch response/data valid bus, store stack response signals, an XI command bus, an XI address bus, and an XI response bus.

This interface was designed to handle one store request per system cycle from each CP, and to optimize the data-return time for CP fetches, while meeting the packaging constraints imposed by the chip technology. Each CP has a dedicated interface, enabling each to transmit fetch, store, and XI operations simultaneously and independently of one another. The XI request controls are separate from those for fetch and store operations, allowing XI requests to be processed regardless of the fetch or store activity for a given CP.

Fetch and store commands are sent from the CP to the L2 via the command and address buses. On store operations, up to one quadword of store data is also sent with the command and address. The information is captured in a CP interface register (the CPAR), and then routed to either the fetch or store controls for further processing, allowing the L2 to accept a new CP operation every cycle.

To optimize CP performance, fetch requests are given high priority and may be routed to prepriority directly from the CPAR. This helps minimize L2 latency on fetch requests. While only one outstanding fetch request per CP is permitted, the L2 does allow the CP to present its next fetch request once the data transfer for the initial request has begun. This, too, is done to minimize storage latency effects.

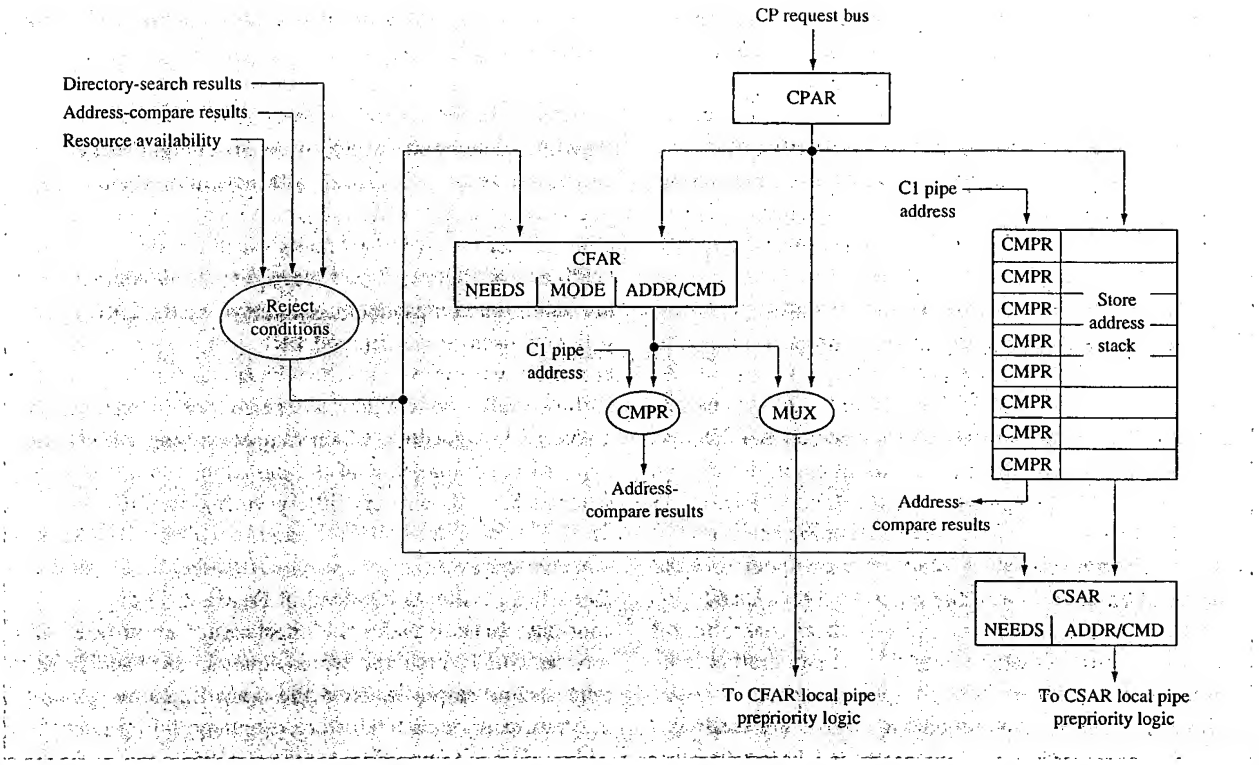


Figure 4

CP controller.

There are three types of fetch requests:

- *Read-only* Typically, these are millicode or S/390 instructions. Read-only data may reside in the L1 caches of multiple CPs.
- *Exclusive* A data line is requested in which one or more bytes will be altered. A CP must have exclusive ownership of a line in order to modify it.
- *Conditional exclusive* This request is usually for operand information. The L2 may grant read-only or exclusive access, depending upon the results of the coherency checking.

Store requests are gated into separate address and data store stacks. The stacks are eight entries deep, allowing the L2 to accept up to eight store requests from each CP. The stack queues the store requests until L2 activity permits the processing of the next sequential store request to the cache.

Since the CP normally owns the bus, each CP is allowed to continue sending store requests unless or until data from a previous fetch request are ready for transfer. When this occurs, the CP controller issues a fetch alert response two cycles before the transfer is to begin. This enables the CP to refrain from further store activity and to switch the BIDI direction to receive the fetch data. Once the data transfer is complete, the BIDI is switched back to its default state, and store request activity can continue.

Cross-invalidate (XI) requests are sent to the CP by the L2 CP controller when the status of a line held in the L1 must change. A line may be invalidated if another CP submits an exclusive fetch request or if the L2 must cast out the line. A line may also be demoted from exclusive to read-only if another CP requests access to the line. An XI request may originate from another CP within the cluster or from another cluster.

Each L2 chip in the cluster can send a new XI request every cycle until the XI stack within the CP is filled. It is the L2's responsibility to track the number of pending XI requests and not overflow the XI stack. As the CP processes the XI requests in FIFO order, it responds by sending XI completion responses to the L2. Queueing the XI requests within the CP helps optimize average XI

latency by enabling a CP to return an XI response every cycle.

Fetch requests

As stated previously, all requests for L2 access must be processed through the L2 pipeline. As a selected fetch request is routed to central priority, the fetch-request controls determine the availability of the desired cache interleave. If an interleave conflict is detected, no pipeline access is permitted in that cycle, effectively blocking lower-priority requestors from "stealing" the pipeline from a fetch request. The fetch request is granted pipeline access the following cycle, and processing continues.

Fetch-request processing may be interrupted once in the pipeline. Each time a fetch request makes a pass through the pipeline, prepriority and central priority calculations are performed. Two mechanisms are used to control the interruption and resumption of fetch requests. These are the NEEDS and MODE registers.

Each CP interface contains a fetch NEEDS register. This register contains an encoded value indicating the condition which must be satisfied in order for the fetch request to resume processing. A NEEDS value of all zeros indicates that all conditions are satisfied; any nonzero value indicates the cause of the interrupted processing. Once the required resource becomes available, the NEEDS register for that CP is reset, and the fetch request is free to reenter prepriority for pipeline access.

Each CP interface also has a MODE register. The MODE is used to "remember" the results of the pipeline pass in progress so that the subsequent passes will be processed more efficiently. For example, checks which were successfully completed on an earlier pass for this request need not be repeated. The NEEDS and MODE registers maximize pipeline efficiency by preventing fetch requests from stalling the L2 pipeline.

To illustrate this concept with an example, suppose CP1 requests a line of data that is currently held "exclusive" by CP2 (another CP on the same cluster). On the first pipeline pass, an invalidate XI request is sent to CP2. CP1's NEEDS is set to indicate that an XI response is required before processing can continue. The MODE register indicates that the request hit in the L2, but to a line owned exclusively by another CP. When the XI response arrives, CP1's NEEDS is reset, and the fetch request makes its second pass through the pipeline. The MODE indicates that the line was held exclusive, so CP2's store stack must now be checked to ensure that the latest copy of the line is identified. If no store to this line is pending, the CP1 fetch request obtains exclusive access to the line and the data are returned. If a store to the requested line is pending, CP1's fetch request is again interrupted, this time to allow the store to be processed. The NEEDS register indicates waiting for stores, and MODE indicates store stack compares detected. Once the cache has been updated with the latest copy of the requested line, NEEDS is reset, and the final pass through the pipeline completes as above [8].

Fetch requests are resolved in one of three scenarios:

1. A fetch can complete in a single pipeline pass. This occurs on requests which hit in the L2 with the requested line held exclusive by the requesting CP; it also occurs on read-only and conditional exclusive requests which hit CP read-only in the L2. The eight requested quadwords are accessed by the cache controls and routed to the CP BIDI data bus. The completion response is sent by the CP controller with the first data transfer, indicating that a new CP fetch request may be sent.
2. A fetch request may make multiple pipeline passes in order to generate local (within this cluster) XI requests. This scenario is in effect if the fetch request hits in the L2 but the line is held exclusive by another CP, or if the request was an exclusive fetch which hits read-only in this L2 and the line is not resident in any other L2 cluster. For these cases, the fetch request makes multiple pipeline passes to resolve the conflict(s) and to update the directory status. On the final pipeline pass, data are returned, and the completion response is sent to the CP.
3. A fetch request results in a request sent on the BSN bus. There are two scenarios in which this occurs. In the first, the request is an exclusive fetch which hits read-only, and the line is in a shared-ownership state where other clusters have copies of the line. All of the copies in other clusters must be invalidated before the current requestor can be granted exclusive access to the line. The BUSRR sends an invalidate request to the remote L2 clusters via the BSN bus and then notifies the requestor upon completion. Once invalidation notification has been received, the requesting fetch is processed through the pipeline as previously described. The second scenario is the L2 miss. This results in an LFAR controller operation to request data from the memory subsystem. In this case, data are returned from one of three sources: main memory, the L2.5, or another L2 cluster. Data are routed to the requestor CP immediately; the LFAR controller then schedules the appropriate cache array update. This approach is used to help minimize the overall L2 miss latency.

Store requests

Store requests from a given CP must be processed in FIFO order. Stores from each of the three CPs compete for access to the pipeline in the prepriority arbitration station. Unlike fetches, store interleave conflicts are checked prior to prepriority. When a conflict is detected, a

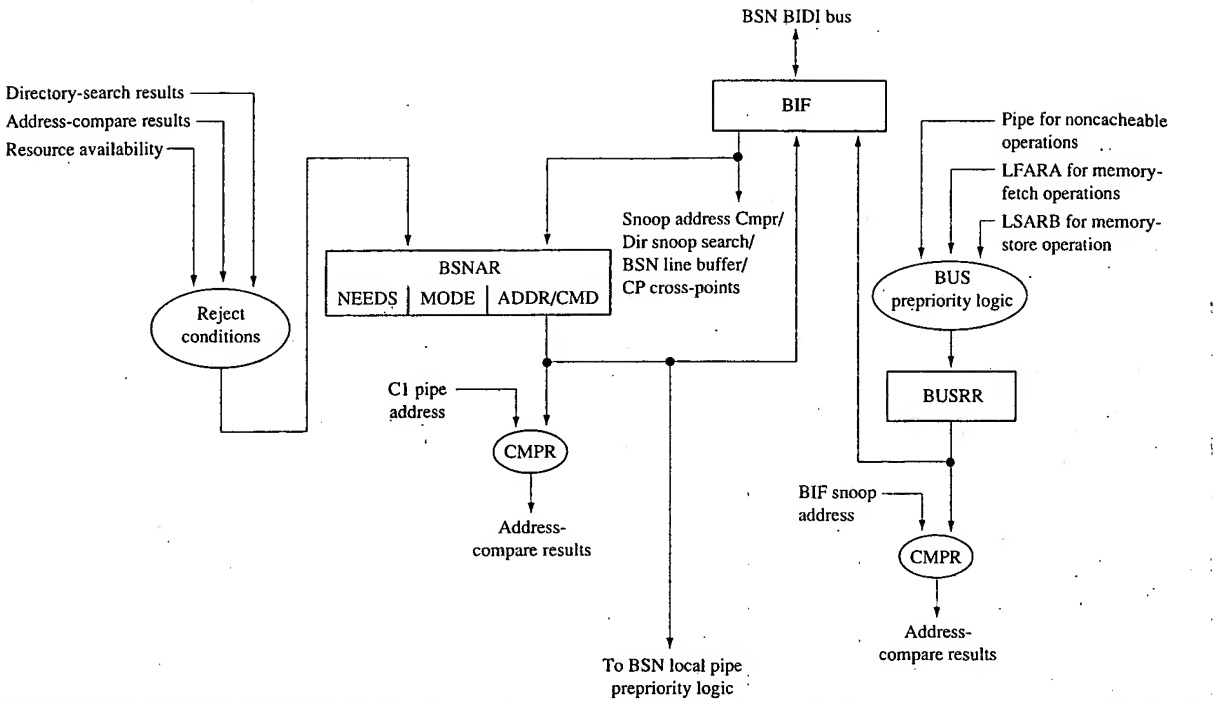


Figure 5

BSN controller.

lower-priority store from a different CP is allowed to proceed. The round-robin algorithm is not updated, however. This allows the first store to have the highest priority on the next cycle, when the interleave conflict should be resolved.

Store requests from a specific CP must be processed in the order in which they are received to ensure coherency. It is possible, however, for a store request to enter prepriority to begin processing before the previous store request from the same CP has completed. If this previous store operation is interrupted, the later store request is also interrupted. Processing on the second store resumes once the initial store operation resumes.

Coherency rules dictate that a CP must have exclusive ownership of a line in order to issue a store request. Therefore, the directory search result must be "hit-exclusive" to the requesting CP. Any other result, with one exception, is an error condition. The exception involves a store request where the directory has not yet been validated for a previous L2 fetch-miss operation (the line may be queued in the LFAR, in the process of updating the cache). In this case, the store request is allowed to continue once the cache and directory have been updated.

- *BSN controller*

There are two BSN controllers on each L2 chip, one dedicated to each BSN port interface. **Figure 5** depicts a single BSN controller. Each BSN controller is divided into two subcontrollers which share a common interface to the BSN bus. The BSNAR is responsible for all bus snoop activity, while the BUSRR is responsible for all bus commands initiated from within a shared L2 cluster. The BSN controller was designed to allow the S/390 G4 L2 design to attach to the S/390 G3 BSN bus, thereby enabling reuse of existing components. The BSN controller therefore contains most of the critical function required to mate the shared L2 cache cluster to the fully shared memory subsystem.

Each BSN port interface consists of a bidirectional bus (referred to as the BSN bus) as well as several unidirectional control signals. The BSN bus is used to transmit command, address, and data. Command and address information is transmitted in the same cycle, while data are sent in subsequent cycles. Because of relatively long memory DRAM access times, the individual memory cards are organized into two independent banks [1]. Consequently, the BSN bus is logically interleaved to take

advantage of this, allowing two separate commands to the memory subsystem to be active simultaneously on each bus (meaning that there can be up to eight active memory operations across the four BSNs in any cycle).

All four shared L2 clusters are considered logically to be connected to the same BSN bus. However, physically there is a separate BSN bus for each shared L2 cluster. When one L2 drives its BSN bus, the BSN chips broadcast a copy on the BSN buses to the other shared L2 clusters on the following system cycle.

Ownership of the BSN bus is established through a straightforward handshake algorithm, using *request*, *grant*, and *command select* control signals. In addition to these control signals, the interface also includes *transfer data* signals to indicate when data are being broadcast on the bus, *memory complete* signals to indicate that a memory operation has completed, and *XI hit* signals to indicate that there is an XI address match on one of the other shared L2 clusters.

BUSRR

All commands destined for the BSN bus which originate in the L2 must pass through the BUSRR controller. The BUSRR register contains a copy of the command and address waiting to be broadcast by this L2. It must pass through the BIF register before being broadcast on the BSN bus. Data to be stored or cast out of the cluster are held in the line store buffer.

The BUSRR can receive command requests from three sources. These sources are, in the priority order in which they are serviced:

1. The pipeline (for example, because of special signaling commands to the CPs).
2. The LSAR controller (an LRU cast-out is required).
3. The LFAR controller (a local CP fetch request results in an L2 cache miss, requiring a snoop to the other clusters).

When the BUSRR receives a command request, it attempts to gain ownership of the BSN bus via a *request* signal. Upon completion of BSN bus arbitration, the BSN bus BIDI is switched to drive mode, and the command, address, and data (if any) are transmitted in subsequent cycles. Once the command and address have been sent, the command is transferred to the BUSRR-B register, freeing up the BUSRR for the next command request. The BUSRR completes the command by notifying the appropriate shared L2 controllers of operation completion, generating encoded responses to be returned to the requesting CP, and controlling the dataflow cross-point to route fetch data to the correct destination.

To reduce L2 miss latency and better manage BSN bus and memory utilization, two performance features have been added to the BUSRR for those cases where a CP fetch misses in the L2 cache:

1. The BUSRR monitors the pipeline in order to predict, up to two cycles in advance, a new fetch request for data not held in the local L2 cluster. If the BSN bus is not currently busy, the BUSRR immediately issues a bus request. Command and address information are then resolved in parallel with bus arbitration, enabling the new request to be transmitted as soon as the bus is available.
2. When fetch data must replace an existing cache entry, the line being replaced must be written back to main memory if it has been modified. Two mechanisms are available—LS/LF [1], and conditional line store (CLST)—for casting out the "old" line with minimal performance impact. The method used is dependent upon the availability of the data being stored relative to the completion of bus arbitration for the fetch request. If the data to be stored are immediately available, the LS/LF operation is used. If not (and this is the more likely case, as an LRU operation may involve XI requests as well), the fetch request is transmitted on the bus immediately. The CLST command is then used to complete the operation, once the bus and the data to be cast out are both available.

BSNAR

The BSNAR is responsible for processing commands originating in the other shared L2 clusters. The incoming command and address are received in the BSNAR via the BIF register, where a directory search is immediately initiated to determine whether the requested data are resident in the L2. If the directory search conflicts with an operation currently in the pipeline, that operation is rejected and scheduled for retry. The BIF-generated directory search must have the highest pipeline priority because of a fixed-cycle relationship on the BSN bus between the request and the activation of *XI hit* to indicate that further action is required in the receiving L2 cluster. In addition to the directory search, additional address compares are performed against the LFAR and LSAR to determine whether the requested data exist in a buffer. This protects data in transition to either the L2 cache or the main memory. This address-compare function is discussed in detail in a later section.

In the event of a directory hit or an LFAR/LSAR address compare match, the *XI hit* signal is raised to notify the requesting cluster that further action is required in the receiving cluster. At this point, the BSNAR requests pipeline prepriority in a manner similar to the other requestors. Like the CP controller, the BSNAR has NEEDS and MODE registers which control and monitor progress through the pipeline.

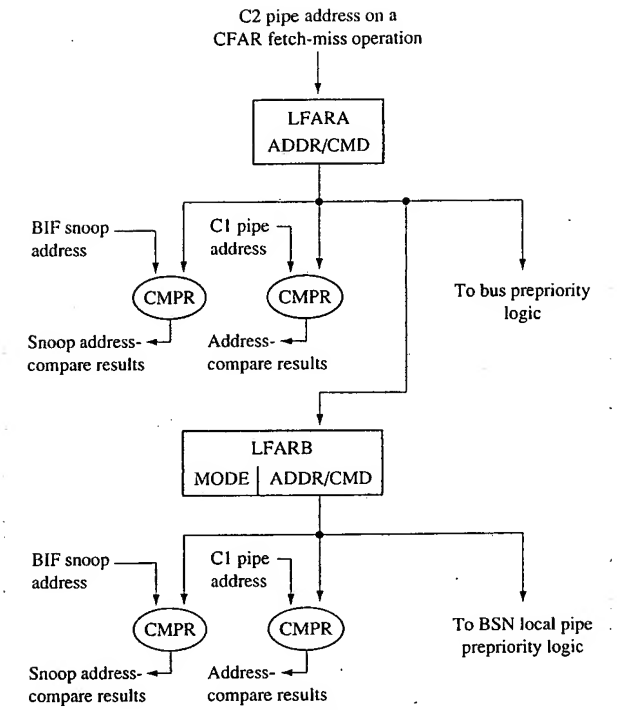


Figure 6

LFAR controller.

The actions that must be performed by the BSNAR are dependent upon the command and the status of the line in question. A whole range of actions are possible, from doing nothing to initiating an XI cast-out to a requesting L2 cluster, with corresponding invalidation of the local cache entry. As an example, suppose the BSNAR receives a fetch-exclusive request from another cluster and that the current status of this requested line is *exclusive to CPx, changed*, where *x* represents any one of the CPs in the address-matched cluster. The BSNAR must

- Send an XI command to CP*x* requesting that CP*x* invalidate its copy of the line.
- Wait for an XI response from CP*x*.
- Wait for possible outstanding store operations to the requested line by CP*x* to complete.
- Access the L2 cache in order to read out the line while updating the directory to invalidate this entry.

Once the line of data has been moved to the line store buffer, the BSNAR must request the BUSRR to issue an XI cast-out with the requested data. The usual handshaking protocols are not needed in this case because the previously raised XI hit ensures that this cluster is the current BSN bus master.

- *LFAR/LSAR controller*

The concept of data buffering has been around for some time; it is an essential design technique for a high-performance multiprocessor system [9]. The basic idea behind buffering is to minimize latency between cache and main storage and also to maximize cache utilization by allowing concurrent cache operations during cache-miss situations. In the application for the S/390 G4 L2, each BSN port contains a line-fetch buffer and a line-store buffer; each buffer also has a pair of associated address registers. The following is a description of buffer-management techniques employed in the LFAR/LSAR controls. Especially noteworthy is the feature which enables each buffer to appear to have the performance of a multiple buffer design. The problem of storage consistency associated with data buffer designs is also addressed.

LFAR

The hardware controller that manages the line-fetch buffer is called LFAR. **Figure 6** depicts this controller. LFAR is used when a request from a CP misses the L2 cache and a BSN bus operation is needed to bring data in from elsewhere in the memory hierarchy. A pair of LFAR registers, LFARA and LFARB, are associated with each BSN port. Thus, there are two LFARA registers and two LFARB registers on the L2 chip.

Because a given L2 may have a single bus operation pending at any point in time (a restriction imposed by the BSN bus protocols), a single buffer was deemed adequate if it could be managed efficiently. A second buffer would have used up valuable chip real estate. Typically, a buffer has only one corresponding address register; in this L2 design, however, a second address register was implemented to enable concurrent fetch-miss operations and to reduce bus idle time between successive fetch operations on the same BSN bus. The LFARA and LFARB registers work in tandem so that while LFARB is waiting for data to return from the BSN bus, LFARA can be loaded with the next outgoing fetch operation. Once the buffer starts clearing out (i.e., the first data shot has been read out to be stored into the cache), the fetch residing in LFARA can initiate a new request to the BSN bus for processing. By the time data return, the data from the previous fetch operation will already have vacated the buffer.

The following describes a fetch-miss operation as handled by the LFARA and LFARB registers. A CP fetch operation detects a miss during its initial pipeline pass (assuming no address or resource conflicts). The fetch address is loaded into LFARA while the victim cache-line

address is loaded into LSARA. A request is made to the BSN for permission to broadcast a fetch command. Once the command is broadcast, the fetch request is moved from LFARA to LFARB to await the return of the data and to start the sequence that will load the data into the L2 cache. Returned data are simultaneously written into the line-fetch buffer and sent to the requesting CP. The LFARB logic requests priority and makes a pass through the pipeline to write the 128 bytes of data from the BSN line buffer into the L2 cache. One quadword is written, per cycle, on consecutive cycles until the entire line is read out of the buffer and written into the cache.
If this process completes without any errors (e.g., no uncorrectable memory errors are detected), another pass through the pipeline is made to validate the directory entry for the new line.

The directory entry validation is performed in a separate pipeline pass for two reasons. First, in the event that an uncorrectable error is detected in memory during the line transfer, the recovery protocol is to post a machine check interrupt code (MCIC) to the operating system, which causes it to deconfigure the physical page frame. Until this occurs, the L2 must block any further references to the corrupted sector of the page frame by not allowing the line to be labeled as valid in the cache. The timing of LFARB's validation pass guarantees that the pass is not made until the last possible storage UE report has been detected. The second reason for having the separate validation pass is to prevent subsequent operations to the line until it is loaded in its entirety. This is accomplished via an address comparator associated with LFARB, which reports an address-compare match in the subsequent operation's initial pipeline pass.

Unlike the CP FAR and the BSN controllers, LFARB is governed only by a mode-state machine register to allow it to transition through three possible hardware states: wait for data, cache load pass, and directory validation pass.

The performance benefit of the second LFAR register is realized because as soon as the cache load from the first buffer entry is underway, the buffer is marked available to receive new fetch data, enabling a new fetch operation to be issued to the BSN bus. With this approach, each L2 is able to achieve concurrent fetch-miss operations with only one data buffer.

As in other controllers in the design, each LFAR has its own address comparator logic for serializing operations that are potentially in conflict with operations currently residing in LFAR. These address comparators serve one basic purpose: They avoid problems relating to data integrity. However, they can easily create deadlock problems if not used carefully. For normal situations, a partial address-compare range comprising the bits used for selecting the directory row (or congruence class address) is sufficient to achieve the desired effect. Although a compare based on a subset of the full line address range results in a compare match overindication, the increase in queueing times caused by it is slight and therefore has a negligible impact on performance. Besides, a compare range based on the directory congruence class was necessary to avoid the situation in which back-to-back fetch misses on the same BSN bus are concurrently targeting the same directory entry for cache-line replacement. A full line address compare is needed to protect the recently acquired storage data from being overlooked on a directory search for a bus snoop operation. A failure to report an XI hit on the BSN bus will violate cache coherency rules.

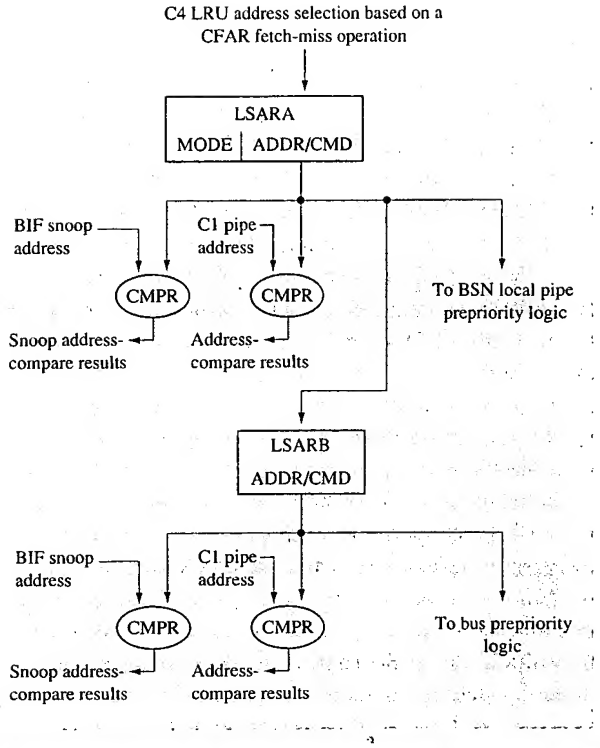


Figure 7

LSAR controller.

LSAR

An LSAR controller is depicted in **Figure 7**. The line-store buffer corresponding to the line-fetch buffer is managed by the line-store address register. Each time a fetch operation from a processor misses the L2 cache, LSAR is responsible for the eviction of the LRU targeted line from the cache to make room for the new fetch-miss data.

In an arrangement analogous to the LFAR just described, there are two LSAR registers, LSARA and

LSARB, dedicated to each BSN port, sharing a single line-store buffer. The cast-out data are held in the buffer until they are allowed to transmit on the bus. Operationally, LSARA has a hardware sequencer (the mode-state machine register) that removes CP ownership of the LRU line and ensures that all possible updates to the LRU line are completed before moving the data from the cache into the line-store buffer. Once data are in the buffer, LSARB continues the cast-out operation by protecting the data (via a full address compare) until the BSN bus is available to process the main memory write-back operation. After the shift from LSARA to LSARB, LSARA becomes available to operate on a new cast-out operation. Just as LFAR is capable of issuing back-to-back fetch operations on the BSN bus, so too can LSAR issue back-to-back write-back operations to main store.

To maintain the full subset cache rule between L1 and L2, each LRU operation requires an invalidation of CP ownership of the cache victim (or LRU) line. In fact, the hardware sequence for invalidating and ensuring the most recent data in the L2 cache is the same sequence used by both the CP and BSN controllers when changing the directory state of an address resident in the L2.

If the LRU line is "dirtied" (meaning that the data are more recent than the main store copy), the data are moved out of the cache and into the line-store buffer to wait until the bus is ready to accept the memory-store operation. During this time, LSARA can start the preparation for a new cast-out, but it will not move the new data into the buffer until any previous cast-out data are on the bus.

Both LSAR registers have a congruence class address comparator to prevent local CP operations from interfering with the LRU congruence class management. This also prevents any inadvertent fetches from being sent to the BSN bus ahead of a cast-out operation, since that would violate cache coherency. An LSARB full address comparator is needed to report an address match in the event that a remote L2 is requesting the data while they are in the line-store buffer.

The cache interactions created by multiprocessors in a shared L2 system require the ability to handle multiple fetch-miss operations even to the same BSN bus. Because of technology constraints, only one line-store buffer per BSN port side was possible, necessitating a high-availability buffer design. This was accomplished by sharing a single line-store buffer between two LSAR registers. The sequence LSARA requires to invalidate the evicted cache line from the local CPs and to ensure that the latest modified data, if any, are inside the cache before moving the data to the line-store buffer can take a number of cycles. By having an LSAR register pair, the L2 is not stalled after an initial fetch-miss operation which ties up both an LFAR and an LSAR. With the second LFAR/LSAR register, a second fetch-miss operation can be started while the initial fetch-miss operation is underway. As soon as that completes, the next fetch operation or cast-out operation is ready to issue a new bus request.

- *Cache array*

The S/390 G4 L2 has a total cache capacity of 384 KB per chip. This is constructed from four physical arrays, each containing 96 KB of data and providing eight bytes per access cycle. Since the L2 chip must provide 16 bytes per cycle per operation, two physical arrays are accessed in parallel to provide the necessary data rate.

Cache accesses that require more than 16 bytes (many accesses are for 128-byte lines) alternate between pairs of physical arrays in successive cycles. Thus, any given physical array is busy only in alternate cycles for multicycle operations. The intermediate cycles are therefore available for use by another operation. This is termed "interleaving," and each pair of physical arrays is referred to as a "cache interleave."

This type of two-way interleaving allows for two simultaneous cache accesses by two independent storage operations. Interleaved operations may be any mixture of fetch or store operations, of any length. If one cache access is already in progress, a second access never has to wait more than one cycle to find the array containing its starting address available for use. In situations where both interleaves are busy, additional storage operations must wait for one of the active operations to complete before access to the cache array is granted.

Each cache interleave has an independent controller that controls all aspects of cache-array access, including address increment and interleave availability notification. For multiple-cycle accesses, addresses and control information are passed back and forth between the interleave controllers on each successive cycle.

Each physical cache array serves both BSN buses. Half of the array addresses are dedicated to one BSN bus, and the other half to the other BSN bus. This restriction exists because of the structure of the cache directories, which is dictated by directory access-time requirements for BSN snoop operations.

The four physical cache arrays are manipulated to appear as two logical arrays (interleaves) with 1024 rows of six compartments each, with 128 bits per compartment.

Cache-array writes may update either 64 or all 128 bits of a compartment to accommodate doubleword and quadword stores from a CP. To store less than a doubleword, the CP first performs a byte merge into a doubleword in the L1 cache, then transfers the entire updated doubleword to the L2 cache.

Figure 8 illustrates the complete dataflow, with cache arrays and including buffers and cross-point switches used

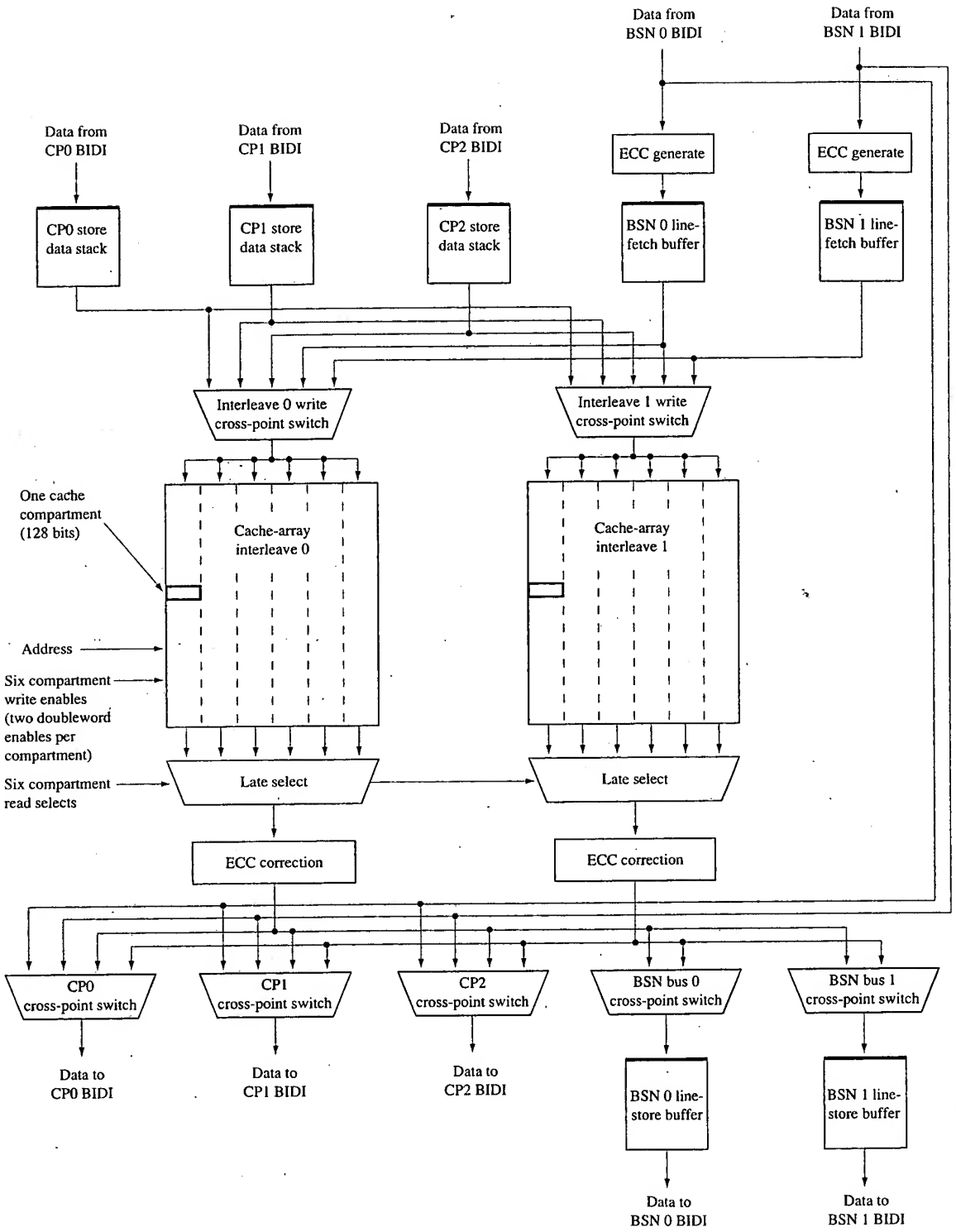


Figure 8

Cache-array structure.

for accessing the cache. This structure of buffers and cross-point switches is designed to maximize system throughput by allowing multiple simultaneous data transfers for different requests.

At the top of the figure, inputs from each of the five bidirectional data buses are shown. At the bottom, outputs are shown to each of these same five bidirectional buses. Each bus may be in either drive or receive mode at any given time, so up to five external data transfers may occur simultaneously.

All data-write paths include a buffer which is used to hold data until the needed cache interleave becomes available for writing. Buffers are needed on the CP interfaces because the CPs are allowed to send a store at any time (provided that the store data stack is not full), and it may be necessary to wait for the cache interleave to complete another operation before the data can be written to the cache. Buffers are needed on the BSN interfaces to allow multiple operations to be active simultaneously. Since the S/390 G4 L2 does not stall waiting for outstanding BSN bus data requests, it is possible that returning data will encounter a busy cache interleave, necessitating the buffer.

All data written to cache must be accompanied by ECC checking bits. The CP stores data with the appropriate ECC check bits already supplied. Data from the BSN bus arrive with parity and are stored with ECC bits generated by the ECC stations (shown).

Data-read paths include buffers for data destined for the BSN bus but not for the CPs. The BSN path requires buffers because when data are being readied for cast-out to the BSN bus, the BUSRR may not yet have been granted priority to send data onto the bus. The fetch-alert response on the CP interface is used to notify the CP in advance that data are about to be returned, blocking the CP from sending any new stores on the BIDI interface, so data that are ready can always be sent immediately to the CP and no buffering is needed.

All data from cache are first checked (and corrected if necessary) by the ECC correction stations (shown), since both the CP and BSN expect to receive data with parity, not ECC.

The seven independent cross-point switches shown in the figure are a very important feature for maintaining performance in the S/390 G4 system SMP environment. These switches operate independently for each CP and BSN port and for each cache array, allowing up to five simultaneous transfers on the five L2 cache chip data ports. Data do not pass through one common bus, which would have the effect of limiting the L2 to one data transfer at a time. Instead, each port is capable of operating independently of the others. Each CP port is capable of receiving data directly from either cache interleave or either BSN bus, each BSN line-store buffer is capable of receiving data from either cache interleave, and each cache interleave is capable of receiving data directly from any CP-store data stack or either BSN line-fetch buffer. As one example, all five L2 chip ports may be active concurrently for the following four simultaneous operations:

1. Fetch from cache interleave 0 to CP0.
2. Fetch from cache interleave 1 to CP1.
3. Fetch from BSN 0 bus to CP2 (busies two L2 ports).
4. Store from BSN 1 line-store buffer to BSN 1 bus.

Shared-cache coherency in the S/390 G4 L2 cache

The S/390 G4 system has three levels of cache hierarchy, in which the first-level (L1) and second-level (L2) caches may contain storage data that are more recent than those existing in the third-level (L2.5) cache or in main memory. The S/390 G4's novel approach to tightly coupling up to 12 microprocessors by creating a system structure consisting of distributed shared-cache clusters led to further novelty in coherence management for a multilevel cache hierarchy. The concept of hierarchical ownership is introduced for the first time in S/390 with the G4 system.

The cache structure for the system consists of store-through L1 caches located on each of the CP chips, shared store-in L2 caches located on separate L2 chips, and store-through L2.5 caches located on each of the four logical BSN buses. Essentially, the L2.5 operates as a main store cache for frequently accessed, shared, read-only data. The L2 caches are supersets of the L1 caches that are part of the same cluster; i.e., if a line of data exists in any of the three L1 caches that are in the cluster, it must also exist in the L2 cache in the same cluster. The reverse is not true, in that a line of data may exist in an L2 cache without existing in any of the L1 caches in the same cluster. However, there is an exception to this subset rule. The storage address range where nonupdatable millicode is kept can be in the L1 while not in the L2. This was done to prevent certain deadlocks on the BSN bus from occurring [10]. There are no superset or subset rules between the L2 and the L2.5 caches, since the L2.5 can never hold data that are more recent than the copy in either the L2 caches or main store.

Coherency is maintained through the use of directory states that are defined as follows for the first two levels of caches. For L1 caches, three directory states are defined: invalid, read-only, and exclusive. The read-only and exclusive states simply tell the CP whether or not the data it has in its L1 can be changed. If the data are in read-only state, a request must be made to the L2 to elevate the state from read-only to exclusive in order to process a store.

For the L2 caches, the following directory states are defined: invalid; read-only to CP(s), MC = 1; read-only to CP(s), MC = 0; and exclusive to a CP. The "MC" in this list refers to a multicopy bit which is used to determine whether the line exists in other shared L2 clusters. A value of 0 indicates that it does not, while a value of 1 indicates that it might.

The L2 directory state is encoded in a three-bit "CPID" field that is part of the directory tag for each cache slot. Since the MC bit is merged into the CPID field, only two bits are used to identify the CP ownership state.

Note that there are unique L2 directory states to keep track of which CP owns a line exclusive, but there are no unique L2 directory states to keep track of which CP or CPs own a line read-only. This was done to allow the limited directory physical space to be used for ECC bits to better protect the directory contents and to reduce design complexity. There is very little resulting loss in system performance.

Cache coherency is primarily managed by each L2 in the system. In this scheme, the L2 simply tells the CP how data it is fetching should be held (read-only or exclusive) in the L1 directory, and also when to invalidate resident data because of another CP's exclusive fetch for the same address or an LRU situation in the L2. There are three compelling reasons for having the coherency management situated in the L2: 1) It simplifies the L1 function; 2) it eliminates L1 directory resource conflict due to bus snoop operations; and 3) it allows the L2 to efficiently manage data in the system. By shielding the L1 from bus snoops, the design avoided having to allow concurrent directory lookups in the L1. In order for the L2 to successfully shield L1 from bus snoops, the L2 must manage the data in its attached L1 caches as a subset of the L2 cache.

With the S/390 G4's system of distributed shared L2 cache clusters, read-only data can be found in one or more of the L2 clusters when the MC status bit is active. When the bit is inactive, the L2 cluster has sole ownership of the data. This MC concept is widely used in bus-based SMP designs, especially in MESI (modified, exclusive, shared, invalidate) schemes. However, because the S/390 G4 system has a multiprocessor node on the system bus, as opposed to a single microprocessor or a private L2, the MC concept and the ownership concept for a shared cache were combined to form a new hierarchical ownership concept applicable only to this type of system structure.

An example demonstrates the advantage of hierarchical ownership: In a bus-based SMP design where each connecting node is a private L2, modified (or "dirtied") data can exist in the L1 and its associated L2 cache. If a CP across the BSN bus requires access to the same data, a data transfer on the bus results, and the data end up being in both L2 caches in a read-only state with the MC bit active. If the CP now wishes to make a modification, a second bus operation is needed to invalidate all multiple copies of the data. On the S/390 G4 structure, where there are multiple CPs in a shared L2 node, if the offending CP is on the same node as the processor that owns the modified data, the fetch operation and the subsequent data transfer are processed entirely within the shared-cache node without requiring the BSN bus. Furthermore, if the offending CP wishes to make a modification, the L2 simply issues an invalidate command to the victim CP, again without requiring the BSN bus. During this entire sequence of events, the MC bit within the L2 remains a 0 while the CPID changes, but only to reflect the ownership changes from exclusive to read-only and back to exclusive states. Obviously, the S/390 G4 structure does not eliminate all CP-to-CP data transfers, but it does reduce the bus traffic and lessen bus queueing, improving system performance.

The L2 chip is also responsible for maintaining strict coherency for both L1 and L2 caches. By strict coherency, it is meant that a CP is allowed to store into a line only if it exists in an exclusive state in its L1 directory and a copy does not exist in any other L1 in the system or in an L2 in any other cluster. Directory states for both levels of caches may have to be updated. This can be accomplished via commands issued by CPs within the cluster or via commands broadcast over the BSN bus from other clusters. For the first type, the CP controller is responsible for making a series of pipeline passes to update the directory states, while for the second type the BSNAR is responsible for making the pipeline passes. Updating of the L1 directory states is done by issuing an XI command to the affected CP(s). Note that for lines that are in a read-only state in the L2 directory, it is not known which CPs, if any, currently own a copy of the line, so XI commands are sent to all three CPs in the cluster. Also note that one advantage of making the L2 chip responsible for both the L1 and L2 directory states is that some of the BSN bus activity is shielded from the CP chips, reducing the number of L1 directory searches that have to be performed.

One interesting case worth mentioning is the case in which a line currently exists in the L2 in a *read-only, MC = 1, unchanged* state and a CP within the cluster makes a request to own the line exclusively. Prior to updating the L1 and L2 directory states within the cluster, the CP controller must first load the BUSRR with a *line invalidate* command to be broadcast on the BSN bus to all other L2 clusters in the system. This forces the other clusters to update their L1 and L2 directory states for this line to invalid. Once the CP controller is notified that the *line invalidate* command is broadcast on the BSN bus, it then proceeds with pipeline passes to update the L1 and L2 directory states within the cluster.

Interlocks and contention management

In a nonblocking switch design such as the S/390 G4 shared L2, where multiple concurrent operations are allowed for improved system throughput, special care must be taken to maintain data consistency. Data integrity is compromised when operations to the same storage data are allowed to overrun, resulting in misplacement of true data and also bad ownership assignments. The S/390 G4 L2 avoids this problem by employing a series of varied address comparators that are observed by each operation as it begins processing but before it can complete. Specifically, these address compares protect against concurrent conflicting data accesses by multiple requestors. And they protect data in transit between L2 clusters and between cache and main memory.

Descriptions of the different address-compare interlocks are listed below. The first section lists the compares observed by the CFAR controller as a request starts up. The second section lists the compares observed by the BSN controller.

• *Interlocks for CFAR requests*

When an L2 receives a fetch request from a local CP, the "valid" bit for that CP's CFAR becomes active immediately, but address conflict must be checked before processing completes. A distinction must be made between a CFAR request which has a conflict and one which has no conflict. The CFAR "valid" tag bit is not sufficient, since it has to be active in both cases, so a special tag called "pending" was created. A request becomes "pending" after it has made an initial directory search pass through the pipeline with no address conflict; i.e., no other requestor is active with the same address. Once a request is "pending," it can proceed safely to completion, as the "pending" tag will block new requests from accessing the same address while it remains active. The tag is reset when the CFAR request is processed.

Different types of address compares observed by CFAR when its request begins are the following:

1. Other CFAR register compares. Each CFAR has a "pending" bit, which indicates that it is valid for compares. If a fetch request sees no compares against other CFARs, the requestor's pending bit is turned on, making this request valid for compares done by other CFARs and effectively locking out other CFAR requests to the same congruence class. The CFAR compare protects against conflicts caused by two CPs attempting to fetch the same line or attempting to fetch lines in the same congruence class. This check avoids the situation in which two CPs attempting to fetch the same line could both see the same directory status (for example, read-only), resulting in an attempt by each to update the line (i.e., trying to obtain exclusive ownership). These types of situations would cause unpredictable directory results.
2. Line-fetch address register (LFAR) compares. This is done to prevent the current request from fetching a line from the same congruence class as a line already being fetched by another CP, thus accessing the same directory and LRU information and potentially overlaying the results of the first request. This check is necessary, since the CFAR for the first request may be reset before the directory update is complete. This is also done to prevent a situation in which two CPs are attempting to fetch the same address which missed the L2. There is a window condition, where the data for the first fetch-miss have returned and are in the process of being loaded into the cache, but the directory entry has not yet been validated. Without the LFAR compare check, the second CP's fetch would detect a directory miss and a new fetch for the same data would be sent out on the BSN bus, resulting in the L2 having two cache slots for the same storage address with potentially different CP ownership.
3. Line-store address register (LSAR) compares. This compare protects against the case in which the current fetch request is for a line that is being LRUed or cast out of the L2.
4. BSN address register (BSNAR) compares. This is done to prevent the current fetch request from attempting to fetch data that are currently being requested by a CP on a remote cluster. To cover the reverse situation, where a CFAR is in the midst of acquiring ownership of some data residing in the L2, any new remote CP request must be held off until the local fetch operation is processed. This interlock is accomplished by having each CFAR maintain another "pending" bit called "BSN pending," which is turned on if no compare is detected against a BSNAR. This makes the CFAR valid for compare for subsequent BSNAR requests.
5. Store-stack compares [8]. These are done to ensure that any outstanding stores to a line are complete before a fetch request for that line is processed, a requirement for maintaining data integrity. Since the store-stack compare is done on a subset of the line address bit range, compares may be indicated when the store is not really to the same line. For this reason, all checking of store-stack compares is gated with a check of the directory hit results. If a CP does not own the line exclusive, it cannot update it. If the line is held exclusive by the requesting CP, compares are checked against the requestor's store stack. If a CP other than the requestor owns the line exclusive, a compare is done against that CP's store stack.

• *Interlocks for BSNAR requests*

The BSN controller is continually "snooping" the BSN bus to determine whether commands being broadcast require action, whether casting out changed data or invalidating old data. When a command is detected, a directory search, using the address that was broadcast on the BSN bus, is initiated. This is done to determine whether the line of data resides in the L2 cache.

In parallel with the directory search, two address compares are done to protect against cases in which the data exist in either the line-fetch or line-store buffer but not in the cache. These two compares are referred to as BIF compares, since the address to which the comparison is made resides in the BIF. These are done using the entire line address bit range.

1. An LFAR BIF compare detects cases where the data are in the line-fetch buffer waiting to be loaded into the L2 cache. If a compare is detected, the BSN controller signals the BSN chip that this L2 has a copy of the line and waits until the LFAR has completed loading the data and updating the cache. Once this occurs, it proceeds normally, i.e., as if a directory hit had been detected on the original directory search.
2. An LSAR BIF compare protects against cases where the line of data resides in the line-store buffer waiting to be broadcast on the BSN bus. If a compare is detected, the BSN controller signals the BSN chip that this L2 has a copy of the line and waits for LSAR to complete its processing of the LRU operation. Assuming that the line was changed in the L2 cache, the data should reside in the line-store buffer at this point, so the BSN controller broadcasts an XI cast-out command on the BSN bus using the data in the line-store buffer, and cancels the line-store command that was loaded into the BUSRR (bus request register) as part of the LSAR LRU operation. If the line was not changed in the L2 cache, LSAR would have completed the LRU operation by simply invalidating the directory entry. In this case, the BSN controller resets without casting out the data, and the data must be fetched from the L2.5 or main store.
3. A BUSRR compare is performed as part of maintaining cache coherency between clusters. When a CP controller detects that the multicopy bit is on while doing a directory search for an exclusive fetch-type command, it loads a *line invalidate* command into the BUSRR register. This command forces other L2 chips to invalidate their copy of the data, so that it can grant exclusivity to the requesting CP. A complication can arise when another L2 broadcasts a command on the BSN bus that accesses the same line of data prior to the *line invalidate* command being broadcast. If the *line invalidate* command is allowed to be broadcast after the other command completes, the directory entries for these data in the different clusters may get out of sync. To prevent this from happening, the BSN controller performs a special BSNAR versus BUSRR compare when processing a command it has received from the BSN bus whenever it reaches a state that causes it to update the local L2 directory state. If a match is detected and the BUSRR register contains a *line invalidate* command, the CP controller is notified to restart the operation from the beginning to see the new L2 directory state and then to take the appropriate course of action to complete the CP request.

Aside from these address-compare checks, when BSNAR goes through the pipeline in its initial pass, it checks for conflicts against pending CP operations. When a CP operation which will result in data returned from the L2 cache is in progress, the BSNAR request must be held up until the new directory state has been updated. This is to ensure that the proper invalidation is sent to the CP which actually owns the data in question. Otherwise a coherency problem will result. The method by which this interlock is accomplished is described in the previous section.

Conclusions

The S/390 G4 shared L2 cache design demonstrates the feasibility and advantages of shared-cache design. It is also evidence that it is possible to create a hybrid design which carries most of the advantages of fully shared cache structures while enabling the use of low-cost system bus topologies. While obviously not as optimal as a true fully shared cache, the shared L2 cache cluster approach provides an attractive low-cost, high-performance alternative. The G4 shared L2 allowed the S/390 design team to maximize reuse of existing componentry without compromising the performance of the S/390 G4 microprocessor. The result is an S/390 G4 CMOS system which matches or exceeds ES/9000 bipolar mainframe system performance at a fraction of the cost.

The basic concepts of this shared L2 cache, most of which were described in this paper, are expected to be capable of being extended to provide greater cache efficiencies and sharing capabilities for future development efforts. For example, the cross-point switches, cache interleaving, data ownership hierarchy, and pipeline arbitration mechanisms can easily be extended to support a larger number of microprocessors in a given cluster. This, combined with expected improvements in CMOS density, can lead to even larger SMP systems or simpler, more powerful SMPs with fewer components. The S/390 L2 cache design team is considering several of these alternatives as we continue to increase S/390 system performance via high-performance shared-cache designs.

Acknowledgments

The successful implementation of this shared L2 cache was in every sense a team effort. The authors wish to thank the additional members of the S/390 L2 logic and verification team: Michael Fee, Carl Ford, Cara Hanson, Kevin Kark, Edward Kaminski, T. C. Lo, Patrick Meaney, A. Rick Seigler, William Shen, Gary Van Huben, and George Wellwood; the members of the S/390 L2 chip integration and test team: William Scarpero, Frank Malgioglio, Adrian Zuckerman, Roy Wood, and Tom Foote; and the S/390 L2 custom SRAM designers led by William Dachtera.

*Trademark or registered trademark of International Business Machines Corporation.

References

1. G. Doettling, K. J. Getzlaff, B. Leppla, W. Lipponer, T. Pflueger, T. Schlipf, D. Schmunkamp, and U. Wille, "S/390 Parallel Enterprise Server Generation 3: A Balanced System and Cache Structure," *IBM J. Res. Develop.* **41,** 405–428 (1997, this issue).
2. K. J. Getzlaff, H. W. Tast, U. Wille, B. Leppla, G. Doettling, W. W. Shen, P. Mak, K. N. Langston, and K. M. Jackson, "Shared Cache Memory Device," filed 7/19/95, U.S. patent pending.
3. C. F. Webb and J. S. Liptay, "A High-Frequency Custom CMOS S/390 Microprocessor," *IBM J. Res. Develop.* **41,** 463–473 (1997, this issue).
4. P. Mak, C. B. Ford, and M. A. Blake, "Computer Architecture Incorporating Processor Clusters and Hierarchical Cache Memories," filed 8/15/96, U.S. patent pending.
5. L. Sigal, J. D. Warnock, B. W. Curran, Y. H. Chan, P. J. Camporese, M. D. Mayo, W. V. Huott, D. R. Knebel, C. T. Chuang, J. P. Eckhardt, and P. T. Wu, "Circuit Design Techniques for the High-Performance CMOS IBM S/390 Parallel Enterprise Server G4 Microprocessor," *IBM J. Res. Develop.* **41,** 489–503 (1997, this issue).
6. B. J. O'Leary and A. J. Sutton, "Dynamic Cache Line Delete," *IBM Tech. Disclosure Bull.*, p. 489 (1989).
7. P. Mak and P. J. Meaney, "Resource Arbitration System with Resource Checking and Lockout," U.S. Patent 5,564,062, 1996.
8. P. Mak, L. J. Clark, S. T. Comfort, C. C. Jones, B. M. Bean, A. E. Bierce, and N. T. Christensen, "Interlock for Controlling Processor Ownership of Pipelined Data for a Store-In Cache," U.S. Patent 5,490,261, 1996.
9. Jim Handy, *The Cache Memory Book*, Academic Press, Inc., San Diego, 1993.
10. P. Mak, W. W. Shen, C. K. Shum, C. F. Webb, S. G. Glassen, and J. R. Easton, "A Mechanism for Locking a Unit of Storage in a Multiprocessor System," filed 3/31/95, U.S. patent pending.

Received January 14, 1997; accepted for publication June 23, 1997

Pak-kin Mak *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (pmak@pk705vma.vnet.ibm.com).* Mr. Mak is a Senior Engineer in the S/390 development organization. He received his B.S.E.E. degree from the Polytechnic Institute of New York and his M.B.A. degree from Union College. Mr. Mak joined IBM Poughkeepsie in 1981, working on the ES/3090 BCE cache design. He has designed high-end system controllers and L2 caches for ES/9021 bipolar-based systems and was the technical team leader for the S/390 G4 shared L2 cache design. Mr. Mak currently holds two patents and has received an IBM Invention Achievement Award, two Outstanding Technical Achievement Awards, and two Division Awards.

Michael A. Blake *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (mblake@vnet.ibm.com).* Mr. Blake is an Advisory Engineer in the S/390 development organization. After receiving a B.S. in electrical engineering from Rensselaer Polytechnic Institute in 1981, he joined IBM in Poughkeepsie, New York, working on logic design in various areas of the ES/3090 and ES/9021 bipolar mainframe systems, including the central processor instruction processing unit, memory controls, and I/O subsystem. Mr. Blake was responsible for the design of the BSN bus interface and controller functions in the S/390 G4 L2 cache. He has received one IBM Division Award.

Christine C. Jones *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (christinej@vnet.ibm.com).* Ms. Jones is an Advisory Engineer in the S/390 development organization. She received a B.S. degree in computer science from the Massachusetts Institute of Technology in 1979. She joined IBM in Poughkeepsie, New York, in 1980, and has contributed to the designs of various shared L2 and system controller functions on ES/3090 and ES/9021 systems. Ms. Jones led the design of the CP controller and cross-point functions in the S/390 G4 L2 cache. She currently holds one patent and has received an IBM Outstanding Innovation Award and an IBM Outstanding Technical Achievement Award.

Gary E. Strait *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (gary_strait@vnet.ibm.com).* Mr. Strait is an Advisory Engineer in the S/390 development organization. He received an A.S. degree in engineering science from Hudson Valley Community College in 1977, a B.S. in electrical engineering from Rensselaer Polytechnic Institute in 1979, and an M.Eng. in electrical engineering from Rensselaer Polytechnic Institute in 1980. He joined IBM in Poughkeepsie, New York, in 1980 and has worked on storage subsystems in the ES/3090 and ES/9021 mainframe systems. Mr. Strait was responsible for aspects of the BSN interface and cache controls. He holds a patent in ECC coding techniques.

Paul R. Turgeon *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (turgeon@pk705vma.vnet.ibm.com).* Mr. Turgeon is a Senior Engineering Manager in the S/390 development organization. After receiving a B.S. in electrical engineering from Rensselaer Polytechnic Institute in 1979, he joined IBM in Kingston, New York, where he was involved with microcode development and RAS design for the 8100 Information System. Mr. Turgeon has held logic design and design management positions in the development of several models of the ES/3090 and ES/9121 mainframe systems. In addition to project management responsibilities for the development of the S/390 G4 L2 cache, Mr. Turgeon was responsible for ensuring that system-level performance objectives were achieved. He currently holds one patent and has received three IBM Division Awards.